Informasjon

Oppgåve	Maksimum poeng	Oppgåvetype
ℹ		Dokument

Digital Design

Oppgåve	Maksimum poeng	Oppgåvetype
1	2	Fleirval
2	3	Fleirval
3	5	Fleirval
4	15	Munnleg
5	5	Fleirval (fleire svar)
6	10	Fleirval (fleire svar)
7	5	Programmering
8	10	Munnleg

Arkitektur

Oppgåve	Maksimum poeng	Oppgåvetype
9	5	Fleirval (fleire svar)
10	5	Fleirval (fleire svar)
11	10	Fyll inn tal
12	5	Fleirval (fleire svar)
13	5	Fleirval (fleire svar)
14	15	Langsvar

ℹ Eksamen IN2060 - Digitalteknikk og datamaskinarkitektur
Eksamensdag: 29. november kl 09:00
Tid for eksamen: 4 timar

Hjelpemiddel: Ingen

Om oppgåvene

Oppgåvesettet består av ulike typar oppgåver, både oppgåver der det vert forventa at du skriv eigne svar og oppgåver med eitt eller fleire riktige val. Nokre oppgåver har fleire felt for svar, og nokon kan ha vedlegg som er vesentlege for å løysa oppgåva. Pass difor på å sjekka at du har lese og svart på heile oppgåva for kvar oppgåve, og nytt gjerne rullefeltene (scrollbar) til å kontrollera at du har fått med deg alt.

Fleirvalsoppgåver med radioknappar kan endrast, men kan ikkje skruast av når du har valt alternativ. Felt med avkrysningsboksar kan ha avgrensing i kor mange kryss som er tillete.

Om poeng i dette eksamenssettet

I dette oppgåvesettet er det mogleg å oppnå inntil 100 poeng totalt. Poenga for kvar oppgåve er oppgjeven på oversynssida for å angje vektingen av kvar oppgåve slik at kandidatane kan disponera tida. Feil i avkrysningsoppgåver kan gje trekk i poeng innenfor oppgåva. Poengtrekk følgjer ikkje med frå ei oppgåve til ei anna. Nøyaktig tal på poeng for kvar riktige eller galne avkrysning vil ikkje bli oppgjevne då dette kan gje svar på kor mange riktige alternativ oppgåva har.

Papiroppgåver

Nokre av oppgåvene må teiknes på eit ark. Pass på at du skriv på kandidatnummer, oppgåvenummer og all anna nødvendig informasjon innafor eksamenstida. Arket eller arka skal leverast inn til eksamensvaktene når eksamen sluttar. Arket vil bli skanna og knytta til den digitale oppgåva i etterkant.

I dette oppgåvesettet kan du svare med digital handteikning (oppgåve 4 og 8). Du bruker skisseark du får utdelt. Du har høve til å bruke fleire ark per oppgåve. Sjå instruksjon for utfylling av skisseark på pult.

Det er IKKJE høve til å bruke digital handteikning på andre oppgåver enn oppgåve 4 og 8. Det blir IKKJE gitt ekstratid for å fylle ut informasjonsboksane på skissearka (eingongskodar, kand.nr. o.l.).

Lykke til!

1 Digital representasjon

Gjer om desimaltalet $(25)_{10}$ til eit 8 bits binærtal.

Vel eitt alternativ

- 00010111
- 00011001
- 00101010
- 00111011
- Ingen av alternativa er korrekte.

Maks poeng: 2

2 Digital representasjon

Gjør om desimaltalet $(-35)_{10}$ til eit 8 bits binærtal på 2'ar kompliment form.

Vel eitt alternativ

- Ingen av alternativa er korrekte.
- 11011101
- 01011110
- 01110111
- 11010011

Maks poeng: 3

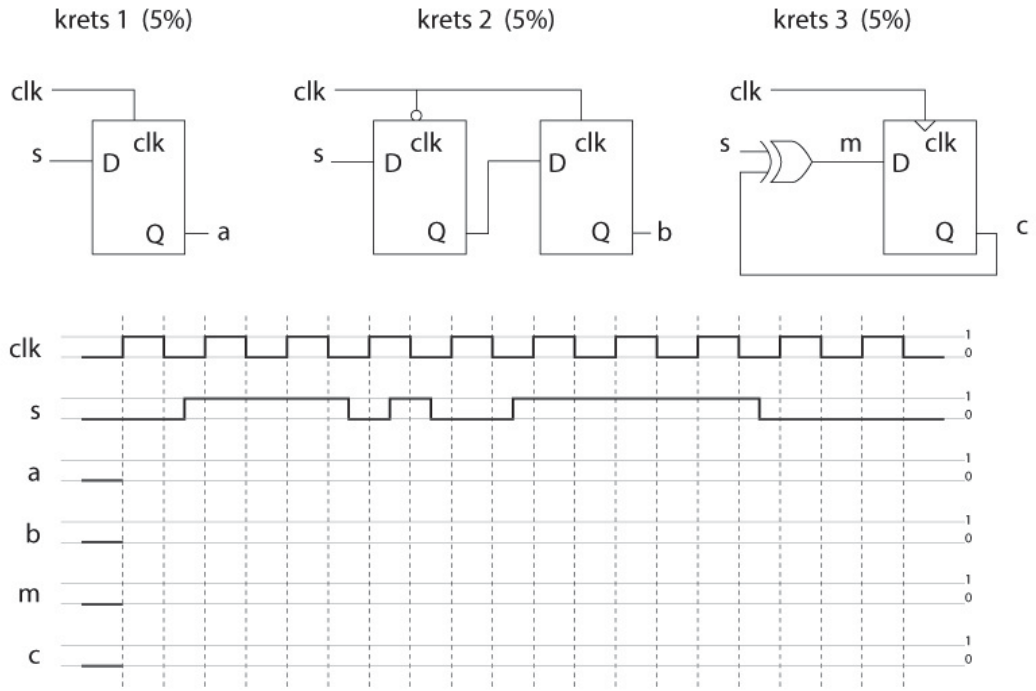
3 Forenkle dette uttrykket maksimalt.
F = XYZ + XY' + XYZ'

Vel eitt alternativ

- Y'+Z
- X
- XY'
- XY
- XY'+Z

Maks poeng: 5

4 Kvar av desse tre kretsane under tek inn klokkesignalet clk og inngangssignalet s. Gå ut frå at a, b, c og m har startverdia 0. Teikn inn det manglande tidsforløpet for signala a, b, m og c. Du skal ikkje ta omsyn til portforseinkinga. Hint: Merk skilnaden på latcher og fliflop i illustrasjonen. **Besvarelsa skal teiknast inn på eit eige ark.**



I dette oppgåvesettet kan du svare med digital handteikning (oppgåve X og X). Du bruker skisseark du får utdelt. Du har høve til å bruke fleire ark per oppgåve. Sjå instruksjon for utfylling av skisseark på pult.

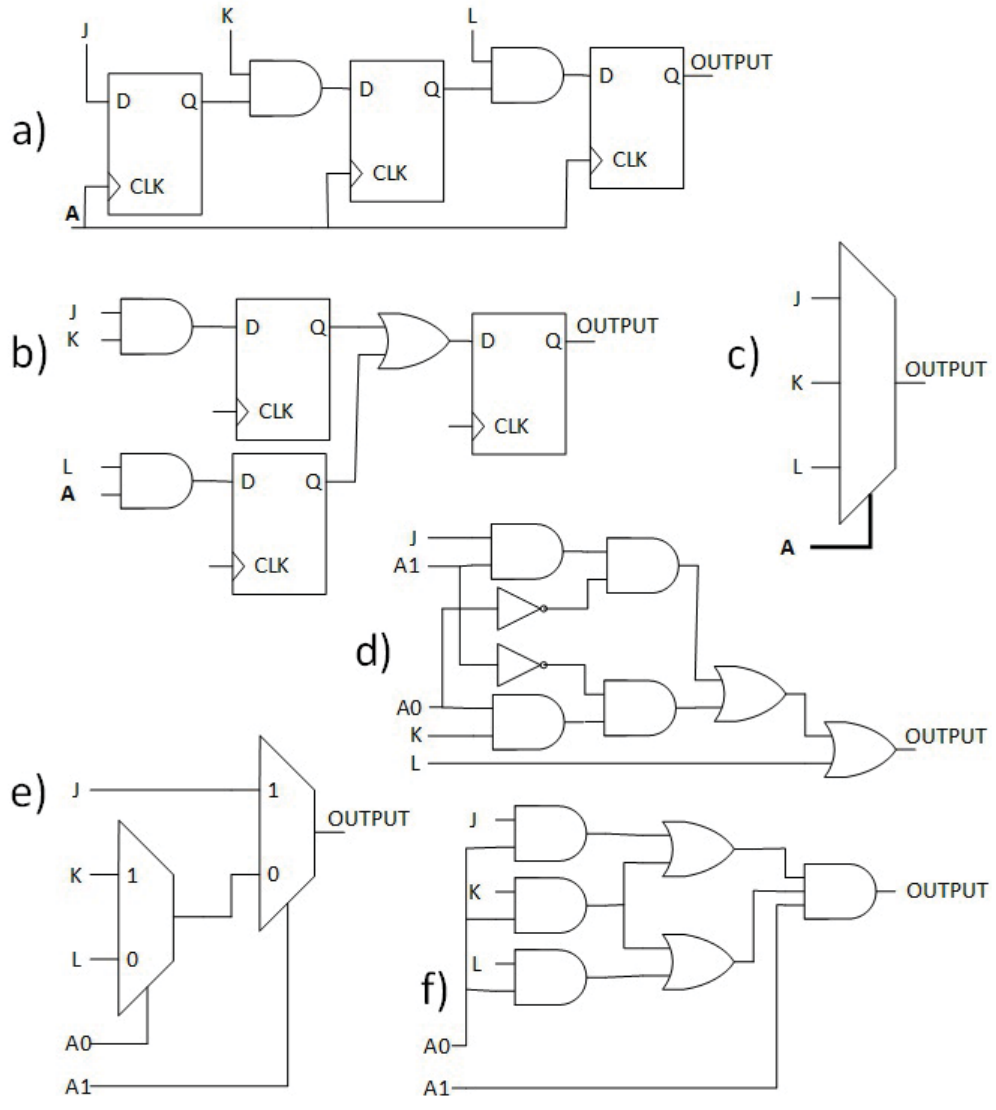
Maks poeng: 15

5

```
library IEEE;
   use IEEE.STD_LOGIC_1164.all;

entity my_thing is
   port( J,K,L  : IN STD_LOGIC;
         A      : IN STD_LOGIC_VECTOR(1 downto 0);
         OUTPUT : OUT STD_LOGIC);
end entity my_thing;

architecture ex of my_thing is
begin
   process(all)
   begin
      if    A(1) = '1' then OUTPUT <= J;
      elsif A(0) = '1' then OUTPUT <= K;
      else                  OUTPUT <= L;
      end if;
   end process;
end architecture ex;
```



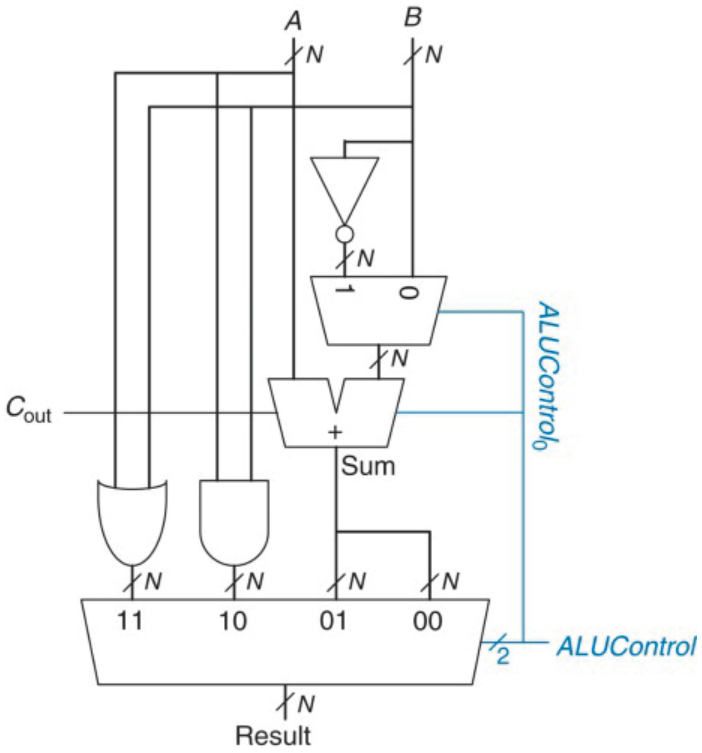
Kva for diagram passar til koden?

Vel eitt eller fleire alternativ

- a)
- b)
- c)
- d)
- e)

Maks poeng: 5

6



Figuren viser ein N-bitar ALU.

Vel eitt eller fleire alternativ

- ALU-en har 3 multipleksarar
- Kontrollsignal 00 vert brukt til addisjon
- C_{out} vil alltid vera '1' når vi legg saman to 2-kompliments tal
- Adderen er av typen Ripple-Carry
- ALU-en har to multipleksarar
- Kontrollsignal 01 må brukast for å leggja saman 2-kompliment tal (signed add)
- Med to slike ALU-ar kan vi multiplisera vilkårlege N-bit tal på to klokkesyklar.
- ALU-en er hjartet i mikrokontrolleren
- Vi kan ikkje utføra A XOR B med denne ALU-en direkte
- Kontrollsignal 10 gjev A AND B
- Nytter vi ALU-en til logiske operasjonar vil C_{out} alltid vera 0
- Kontrollsignal 10 gjev A OR B

Maks poeng: 10

7

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity decoder2_4 is
  port(a: in STD_LOGIC_VECTOR(1 downto 0);
       y: out STD_LOGIC_VECTOR(3 downto 0));
end entity;

architecture synth of decoder2_4 is
begin
   process(all) begin
      case a is
         when "00" => y <= "0001";
         when "01" => y <= "0010";
         when "10" => y <= "0100";
         when "11" => y <= "1000";
         when others => y <= "0000";
      end case;
   end process;
end architecture;
```

Over ser du fullstendig VHDL-kode til ein 2 til 4 dekodar.
Lag fullstendig VHDL kode til ein 3 til 8 dekodar.
Kopier og modifiser gjerne frå koden over.

Skriv svaret ditt her...

Maks poeng: 5

8

```
library IEEE;
  use IEEE.STD_LOGIC_1164.all;

entity fsm is
  port( CLK, RESET : IN  STD_LOGIC;
        A, B       : IN  STD_LOGIC;
        P, Q, R    : OUT STD_LOGIC);
end entity fsm;
```

```
process(all)
  begin
    case state is
      when S0 =>
        if ((A = '1') or (B='1')) then nextstate <= S0;
        else                           nextstate <= S1;
        end if;
      when S1 =>
        if    (A= '1')     then nextstate <= S0;
        elsif (B= '1')     then nextstate <= S1;
```

```
architecture ex of fsm is
begin
  type statetype is (S0, S1, S2);
  signal state, nextstate : statetype;

process(CLK, RESET)
  begin
    if RESET then state <= S0;
    elsif rising_edge(CLK) then
      state <= nextstate;
  end if;
end process;
```

```
      else                    nextstate <= S2;
      end if;
    when S2 =>
      if (A='1')      then nextstate <= S1;
      elsif (B='1')   then nextstate <= S2;
      else                 nextstate <= S0;
    end if;
  when others => nextstate <= S0;
end process;

-- concurrent statements
P = '1' when state = S0 else '0';
Q = '1' when state = S1 else '0';
R = '1' when state = S2 else 'Z';

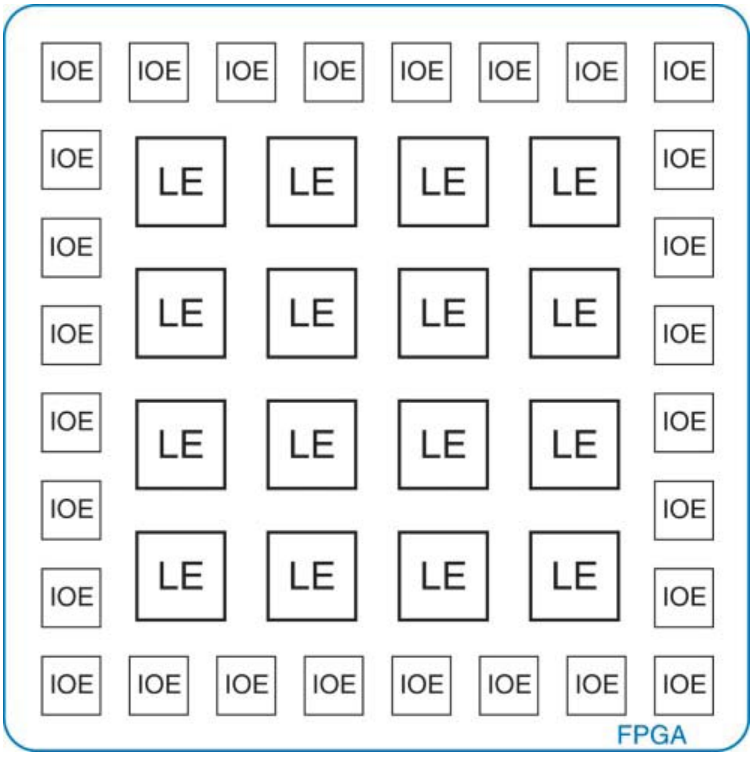
end architecture ex;
```

Teikn eit tilstandsdiagram for tilstandsmaskinen beskriven med VHDL-koden (over) på eit eige ark.

I dette oppgåvesettet kan du svare med digital handteikning (oppgåve X og X). Du bruker skisseark du får utdelt. Du har høve til å bruke fleire ark per oppgåve. Sjå instruksjon for utfylling av skisseark på pult.

Maks poeng: 10

9



Figuren over viser ein generell arkitektur for ein FPGA bestående av logiske element (LE) og input-output element (IOE).

Vel eitt eller fleire alternativ

- FPGA kan verte nytta til både kombinatorisk og sekvensiell logikk
- Ein FPGA kan ikkje innehalda RAM
- Logiske element kan innehalda oppslagstabellar (LUT-ar)
- Når vi programmerer ein FPGA avgjer vi kva for delar av dei logiske elementa som vert nytta og korleis desse vert kopla saman
- IOE-blokkar kan ikkje ha tristate-buffer
- Tristate-buffer vert gjerne brukt til bussar med fleire drivarar
- Når vi programmerer ein FPGA med VHDL skjer instruksjonane sekvensielt
- Avstanden mellom logiske element som vert kopla saman er ubetydeleg for frekvensen klokka kan ha
- Det er lurt å unngå bruk av typen INOUT for interne signal, fordi det kan føra til unødig bruk av IOE-blokkar

Maks poeng: 5

10

```
.text
.global main
main:
        MOV r5, #10
        MOV r7, lr
        BL add_one
        BX lr

add_one:
        ADD r0, r5, #1
        BX lr
```

Vel eitt eller fleire alternativ

- 'main' nytter feil register til funksjonargument
- 'main' lagrar link-registeret riktig før funksjonskall
- 'add_one' returnerer verdi i feil register
- Programmet vil aldri avslutta

Maks poeng: 5

11 Vi ynskjer å omsetja følgjande program til ARM assembler. Du kan anta at 'g' ligg i 'R0' og 'h' ligg i 'R1'.

```
if(g < h) {
    h = h + 1;
} else {
    h = h * 2;
}
```

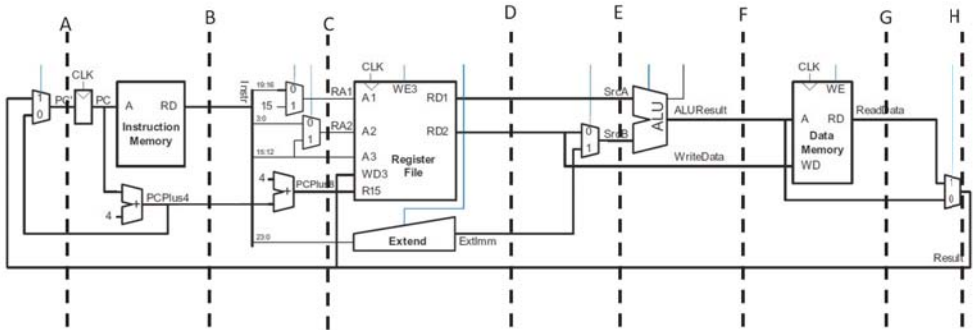
Svar på følgjande spørsmål om mengd instuksjoner i den omsette assembler koden:

a. Viss ein berre kan nytta betingakøyring (*conditional execution*) på hopp (*branch*), treng ein minimum [ ] instruksjoner.

b. Viss ein kan nytta betingakøyring (*conditional execution*) på alle instuksjoner treng ein minimum [ ] instruksjoner.

Maks poeng: 10

12 På kva for stadar er det best å setja inn pipeline-register for å få ein 5-steg pipeline prosessor?



Vel eitt eller fleire alternativ

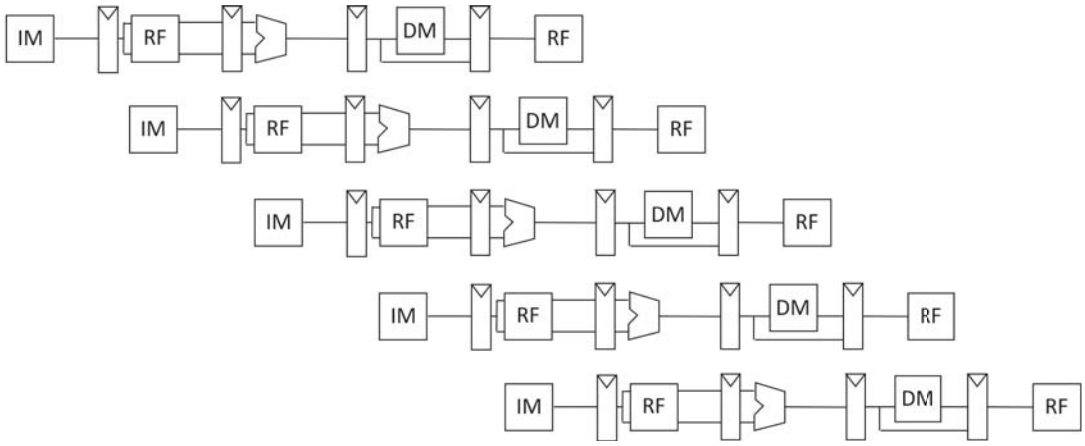
- [ ] CEFH
- [ ] BDFG
- [ ] BCDG
- [ ] ABCEF
- [ ] ACDFG
- [ ] ACEFH

Maks poeng: 5

13 Gjeven ARM-assemblerprogrammet:

```
MOV R3, #28
SUB R0, R3, #3
LDR R4, [R0, #14]
STR R5, [R3, #36]
AND R2, R0, R4
```

Kva for (eit) register vert skrive og / eller vert lese i sykel 5?
Anta 5-steg pipelinet prosessor med hasardeining som i boka.
Du kan bruka illustrasjonen som tankehjelp / for å kladda.

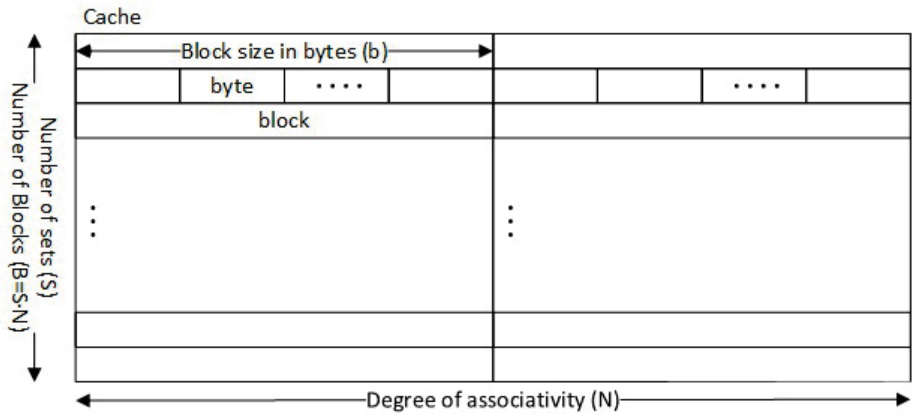


Vel eitt eller fleire alternativ

- R2
- R4
- R3
- R0
- R5

Maks poeng: 5

14



Figur av ein generisk cache.

Vi har fylgjande ARM assembler kode:

```
(1)              MOV  R0, #5
(2)              MOV  R1, #0
(3)  LOOP        CMP  R0, 0
(4)              BEQ  DONE
(5)              LDR  R2, [R1, #0x4]
(6)              LDR  R3, [R1, #0x8]
(7)              LDR  R4, [R1, #0x24]
(8)              SUB  R0, R0, #1
(9)              B    LOOP
(10) DONE
```

I denne oppgåva skal du samanlikne kor godt to ulike cachesystemer yter:
1) Ein direkte kobla (direct mapped) cache med blokkar bestående av fire ord.
2) Ein 2-way set associative cache bestående av blokkar på eitt ord.
Både cachene nyttar ord på 4 byte og 8 sett.

a) Kva blir miss-rate for cache 1) og 2) ? (Skriv som brøk)

Vi endrar linje 6 til

```
(6) LDR   R3, [R1, #0x88]
```

b) Vil miss-ratane for dei to cachene endre seg med den nye koden? Grunngje svaret.

For kvar av cachene innfører vi ein nivå 2 cache (level 2) med 16 gonger så mange sett (128 sett), men same oppbygning som 1) og 2).
Vi antek at miss penalty er på 100 klokkesyklar om vi må lese frå RAM og 10 klokkesykler om vi må lese frå level 2 cache.

c) I kva for grad vil cachesystema 1) og 2) kunne dra nytte av nivå-2-cachen, om vi benytter den same assemblerkoden som i b)? Grunngje svaret.

Skriv svaret ditt her...

Format | B I U x_2 x^2 | I_x | | | | | | Ω | | | ∑ |

Words: 0

Maks poeng: 15

Data-processing instructions

Name	Description	Operation
`ADD Rd, Rn, Src2`	Add (`+`)	`Rd = Rn + Src2`
`SUB Rd, Rn, Src2`	Subtract (`-`)	`Rd = Rn - Src2`
`AND Rd, Rn, Src2`	Bitwise AND (`&`)	`Rd = Rn & Src2`
`ORR Rd, Rn, Src2`	Bitwise OR (`|`)	`Rd = Rn | Src2`
`EOR Rd, Rn, Src2`	Bitwise Exclusive OR (`^`)	`Rd = Rn ^ Src2`
`BIC Rd, Rn, Src2`	Bitwise Clear	`Rd = Rn & ~Src2`
`MVN Rd, Rn, Src2`	Bitwise NOT (`~`)	`Rd = ~Rn`
`LSL Rd, Rn, Src2`	Logical Shift Left (`<<`)	`Rd = Rn << Src2`
`LSR Rd, Rn, Src2`	Logical Shift Right (`>>`)	`Rd = Rn >> Src2`
`MOV Rd, Src2`	Move (`=`)	`Rd = Src2`
`CMP Rd, Src2`	Compare	Set flags (see below) based on `Rd - Src2`

Remember that we can also set condition flags by appending an `S` to the end of our Data-processing instructions.

Name	Description
`ADDS Rd, Rn, Src2`	Add (as above) **and** set condition flags
`SUBS Rd, Rn, Src2`	Subtract (as above) **and** set condition flags
`ANDS Rd, Rn, Src2`	Bitwise AND (as above) **and** set condition flags

Multiply instructions

Name	Description	Operation
`MUL Rd, Rn, Rm`	Multiply (`*`)	`Rd = Rn * Rm`
`MULS Rd, Rn, Rm`	Multiply (`*`) **and** set condition flags	`Rd = Rn * Rm`
`MLA Rd, Rn, Rm, Ra`	Multiply and Accumulate	`Rd = (Rn * Rm) + Ra`

Memory instructions

Name	Description	Operation
`STR Rd, [Rn, ± Src2]`	Store Register	`Mem[Adr] = Rd`
`LDR Rd, [Rn, ± Src2]`	Load Register	`Rd = Mem[Adr]`

Branch instructions

Name	Description	Operation
`B label`	Branch	`PC = (PC + 8) + imm24 << 2`
`BL label`	Branch and Link	`LR = (PC + 8) - 4; PC = (PC + 8) + im24 << 2`
`BX Rd`	Branch and eXchange	Branch to address pointed to in `Rd` (used for return)

Condition flags

Flag	Name	Description
`N`	Negative	Instruction result is negative
`Z`	Zero	Instruction result is zero
`C`	Carry	Instruction caused a carry out
`V`	oVerflow	Instruction caused an overflow

Condition mnemonics

Mnemonic	Name	CondEx
`EQ`	Equal	`Z`
`NE`	Not Equal	`!Z`
`CS/HS`	Carry set / unsigned higher or same	`C`
`CC/LO`	Carry clear / unsigned lower	`!C`
`MI`	Minus / negative	`N`
`PL`	Plus / Positive *or* zero	`!N`
`VS`	Overflow	`V`
`VC`	No overflow	`!V`
`HI`	Unsigned higher	`!Z AND C`
`LS`	Unsigned lower or same	`Z OR !C`
`GE`	Signed greater than or equal	`!N XOR !V`
`LT`	Signed less than	`N XOR V`
`GT`	Signed greater than	`!Z AND (!N XOR !V)`
`LE`	Signed less than or equal	`Z OR (N XOR V)`

Theorems

Number	Theorem	Dual	Name
T1	B • 1 = B	B + 0 = B	Identity
T2	B • 0 = 0	B + 1 = 1	Null Element
T3	B • B = B	B + B = B	Idempotency
T4	(B')' = B		Involution
T5	B • B' = 0	B + B' = 1	Complements

#	Theorem	Dual	Name
T6	B•C = C•B	B+C = C+B	Commutativity
T7	(B•C) • D = B • (C•D)	(B + C) + D = B + (C + D)	Associativity
T8	B • (C + D) = (B•C) + (B•D)	B + (C•D) = (B+C) (B+D)	Distributivity
T9	B • (B+C) = B	B + (B•C) = B	Covering
T10	(B•C) + (B•$\overline{C}$) = B	(B+C) • (B+$\overline{C}$) = B	Combining
T11	(B•C) + ($\overline{B}$•D) + (C•D) = (B•C) + ($\overline{B}$•D)	(B+C) • ($\overline{B}$+D) • (C+D) = (B+C) • ($\overline{B}$+D)	Consensus

#	Theorem	Dual	Name
T12	$\overline{B_0 \bullet B_1 \bullet B_2 \ldots} = \overline{B_0} + \overline{B_1} + \overline{B_2} \ldots$	$\overline{B_0 + B_1 + B_2 \ldots} = \overline{B_0} \bullet \overline{B_1} \bullet \overline{B_2} \ldots$	DeMorgan's Theorem